User's Guide

Authors

- Prof. Alberto Leva
- Eng. Davide Cavazzini
- Eng. Riccardo Schiavi
- Eng. Davide Aliprandi
- Eng. Filippo Donida
- Eng. Gianmario Fontana
- Eng. Stefano Iacinti
- Eng. Alex Mancastroppa
- Eng. Paolo Romagnoli
- Eng. Fabio Arata
- Eng. Ivano Brambati
- Eng. Emanuele Bulgarini
- Eng. Miriam Corini

Getting Started

Install Java, OMC and SimForge according to the Installation Notes.

Important: when launching SimForge for the first time, go to Tools|Settings, and set the paths.

On **Windows** machines:

- OPENMODELICAHOME contains the OMC root directory, e.g., `C:\OpenModelica1.4.4`
- OPENMODELICALIBRARY points to the directory which contains the `Modelica` directory of your favourite Modelica Standard Library package; you can use the one provided with the OMC installation, e.g., `C:\OpenModelica1.4.4\ModelicaLibrary`
- LD_LIBRARY_PATH is irrelevant

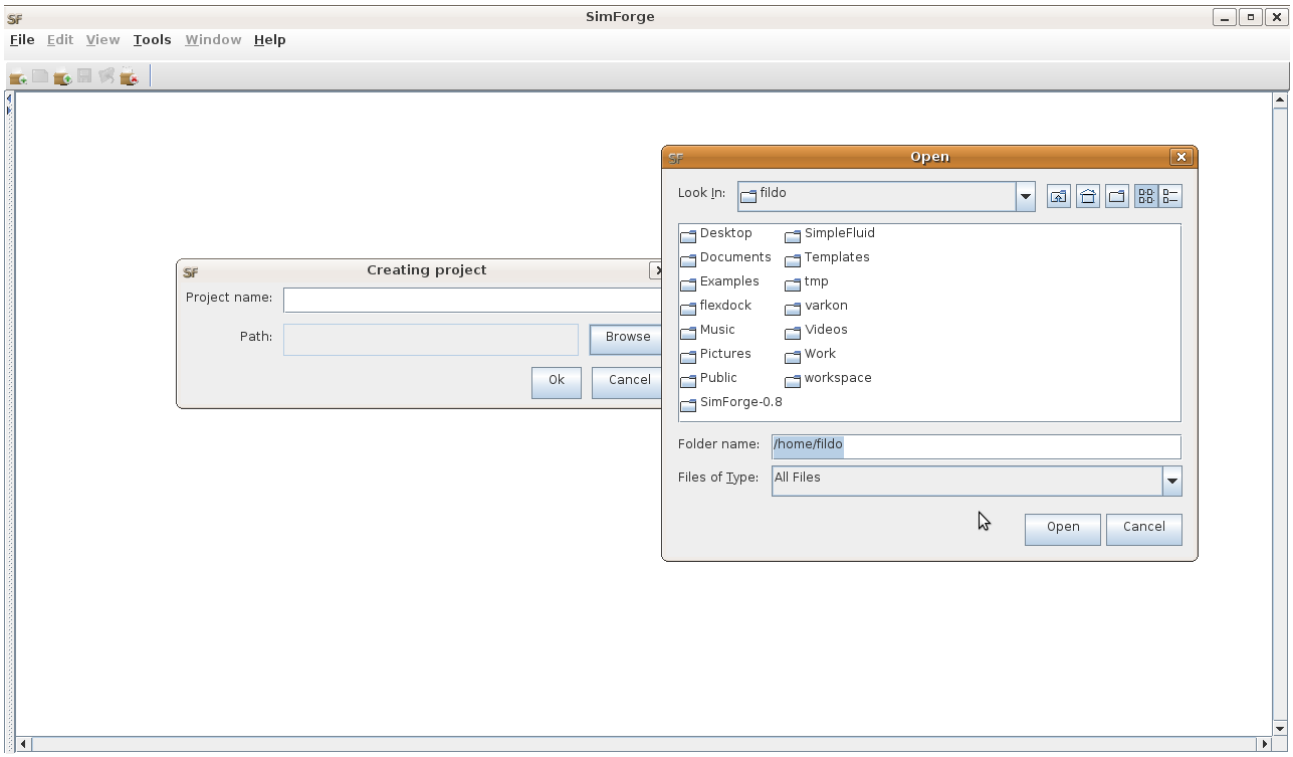
On **linux** machines:

- OPENMODELICAHOME must point to the directory which contains the `bin` subdirectory where the `omc` executable is located, e.g., `/usr/share/openmodelica-1.4/`
- OPENMODELICALIBRARY points to the directory which contains the `Modelica` directory of your favourite Modelica Standard Library package, e.g., `/usr/share/openmodelica-1.4/modelica-2.2/`
- LD_LIBRARY_PATH points to the directory containing the `libmico` library; you can use the one provided by your OMC installation, e.g. `/usr/lib/mico-openmodelica-1.4`

These will be the default choices for all the new projects which are created. They can be overridden within each project by using project-specific settings.

Creating a new project

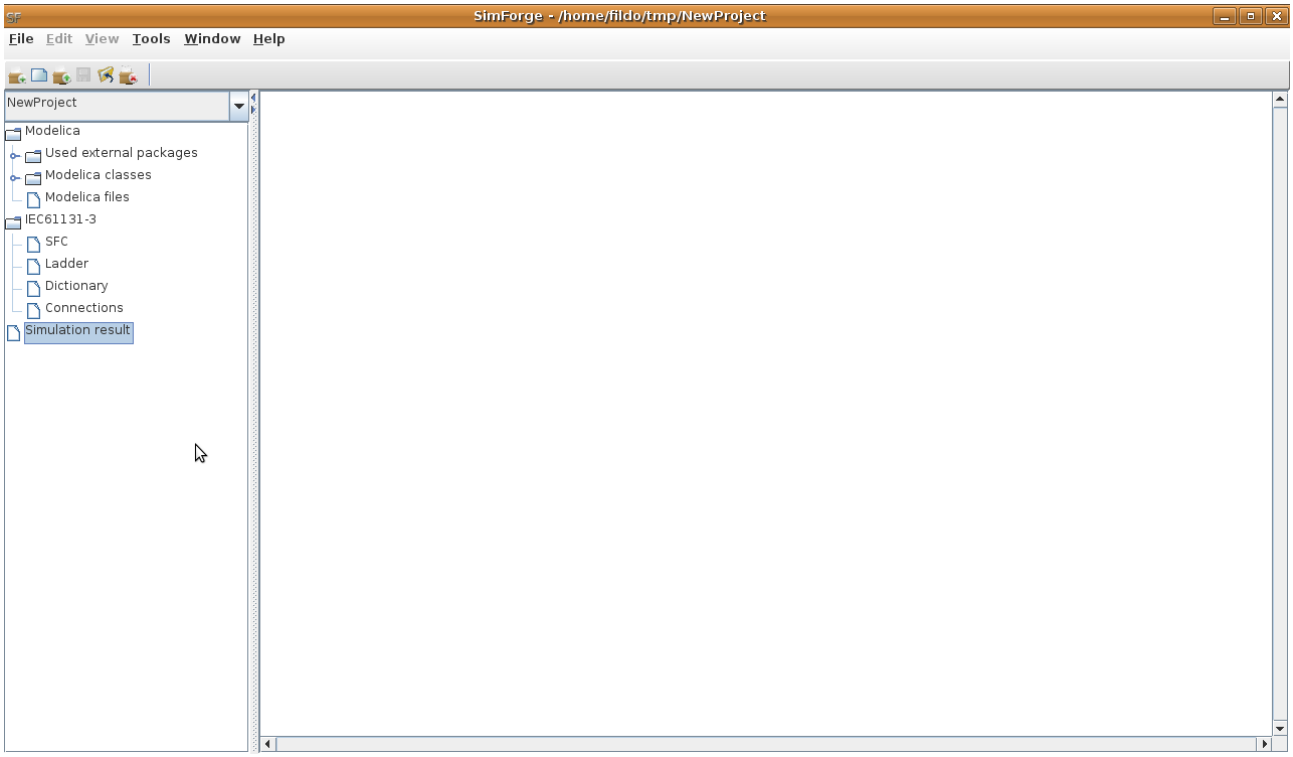
Go to File|New project. Choose a name for the project - this will be the name of the directory containing all project files, so it must be a legal directory name - and select the path where you want this new directory to be created.

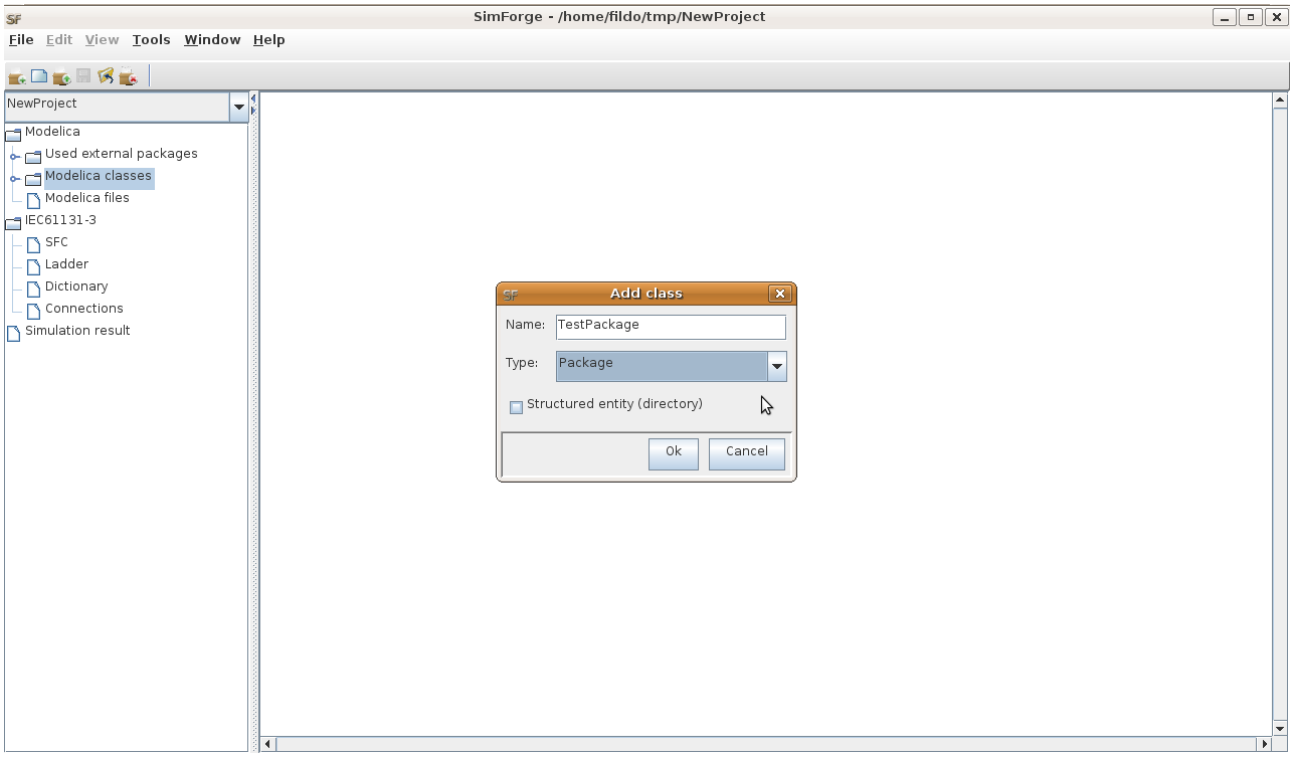


SimForge will create the project directory, which contains: the IEC61131 directory, for IEC 61131 controllers code, the Modelica directory, for Modelica code, the Results directory, for permanent storage of simulation results, the Temp directory for temporary files, and the Properties.xml file for the project properties.

Double-click on the Modelica node of the project tree. The three nodes contain the following elements:

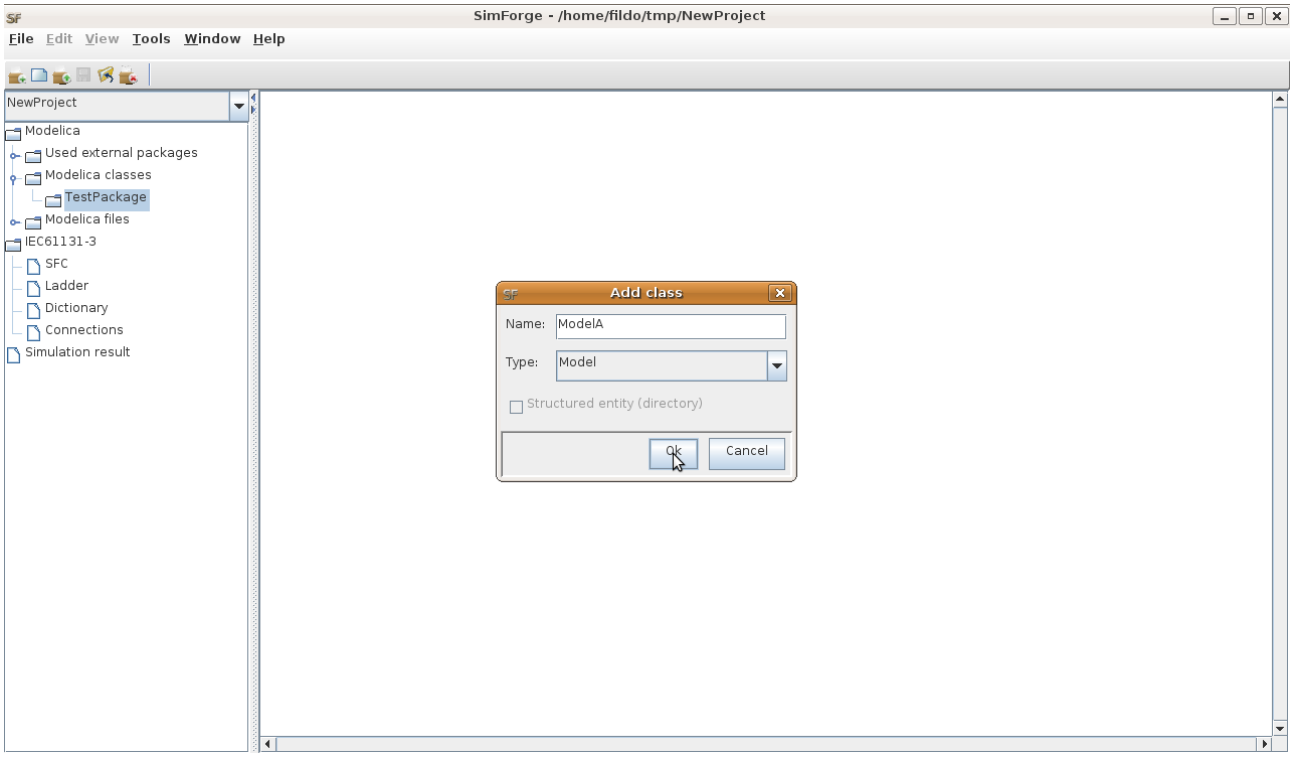
- *Used external packages*: all the packages which are used by the project, but which are not part of the project itself, so they are read-only, and are not contained within the project directory. Examples: the Modelica Standard Library, or any other free or commercial third-party library that will be used by the current project.
- *Modelica classes*: a tree view of all the models and packages defined within the project, irrespective of their actual representation as single files or structured entities (i.e., sub-directories). Models within this tree can be edited in both graphics and textual modes.
- *Modelica files*: a view of all the .mo files contained in the project. Can be useful to inspect files, and rearrange the file contents. Text-only editing.



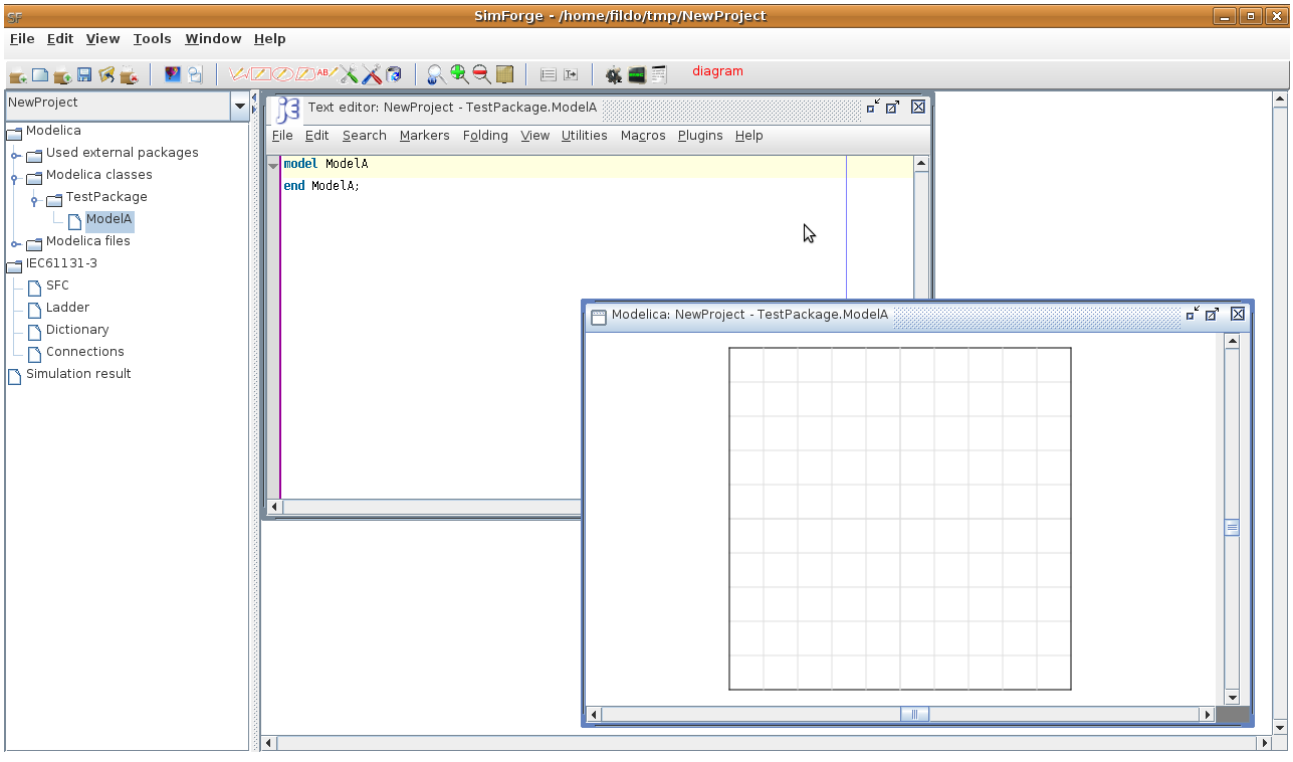


At this point you can create one or more top-level models or packages, which will be saved in the Modelica subdirectory of the project. For instance, right-click on Modelica classes and type TestPackage as the package name, Package as the class type. If you leave 'structured entity' unchecked, the package will be saved as a single .mo file, otherwise it will be saved as a sub-directory containing a package.mo file plus files for all other contained classes.

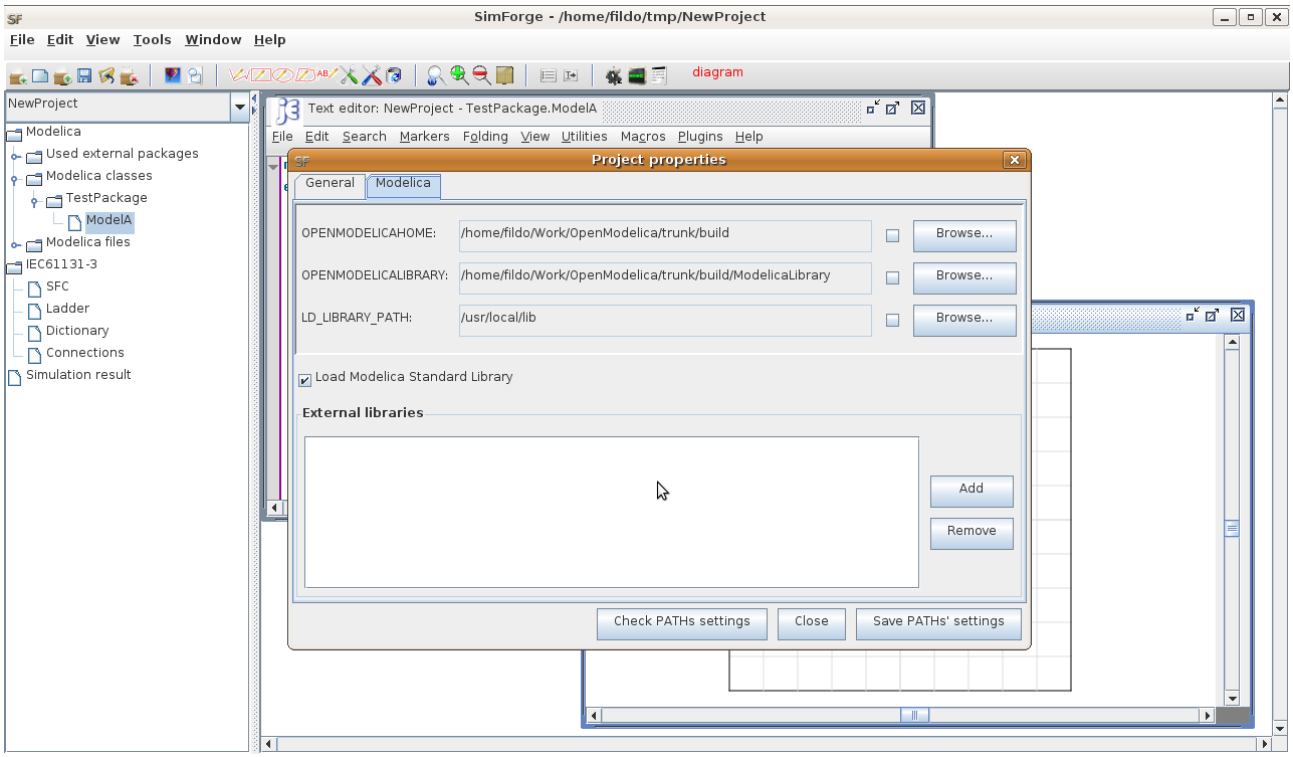
Now right-click on TestPackage in the project tree, select 'Add class', then type in *ModelA* and select *model* as the type of class.



Double-click on the ModelA node to open the text and graphical editors. You can now start editing your model. The three icons on the right of the tool bar can be used to check the model, compile and simulate it, and to show the omc console.



You can now customize the project properties. Click on Tools|Project properties. The General tab provides information about the project's path, as well as some user-defined comments to the project.



The Modelica tab contain the following settings:

- The OPENMODELICAHOME, OPENMODELICALIBRARY, and LD_LIBRARY_PATH paths for this project. By default these are set to the tool settings, but it is possible to customize them by clicking on the checkbox and providing a new path, so that each project can use its own version of OMC, Modelica Standard Library, and OMC runtime libraries.
- A checkbox to load the Modelica Standard Library. If you want to use the Modelica Standard Library for your project, you have to set this checkbox explicitly when you first create the project, and make sure you then click on the *Reload omc* button, so that omc loads the standard library, and makes it visible immediately in the project tree. This is not necessary when you later re-open the project, as the required Modelica library will be loaded automatically.
- Additional external libraries: you can add them here by giving the paths of the packages. They will show up under the *Used external packages* node of the project tree, as read-only classes.

Note that it is possible to load multiple projects at the same time in SimForge. They can be selected by the drop-down menu on top of the project tree. Each project talks to a separate instance of OMC, so it is possible to open different projects using different versions of OMC and standard library at the same time within a single instance of the SimForge application. It is also possible to load two different projects that contain two different versions of the same package (thus with the same name). All the class editing windows contain information about the project name, so that there's no chance of being confused.

Editing models and classes

To edit models (or generic classes), right-click on the corresponding node under the Modelica classes tree and select Open class, or just double-click the node if it is a model (double-clicking on package nodes opens and closes the node). You can type in code in the text window, which offers on-the-fly syntax highlighting. Please note that the current verion of SimForge does not preserve whitespace - this is not a bug of SimForge, but is a problem with OMC: each time the text window content is passed to OMC, it gets parsed, and some whitespace information is lost or modified. Future releases of OMC will include a separate parser to preserve this additional information (see this paper), and this will automatically improve the behaviour of SimForge.

Alternatively, you can drag and drop components from the project tree into the diagram layer. Components can be moved by just dragging them around. Multiple selection is supported, by drawing a rectangle with the mouse (which will select all the included components), or by pressing Ctrl-click on each component. To resize a component, grab it on the edges and drag them.

When you pass with the cursor over a connector and stop for a few seconds, the name of the connector pops up. In order to make a connection, *keep the shift key pressed*, *click and release* the mouse button on the connector, then move the cursor to the other connector and click again. Please note that dragging the cursor before first releasing the mouse button voids the connection.

When working on the graphical layers, you can switch between the icon layer and the diagram layer using either the two buttons on the toolbar, or the View menu. The positions of outer connectors in the diagram and icon view are independent, according to the Modelica 3.0 graphical annotations, as well as the icon and diagram extents. To set the extent, click on the *Class properties* button in the tool bar. You can draw on the icon and diagram layer using the drawing tools on the toolbar.

To set parameters, modifiers, attributes and graphical properties of a components, select it with the left mouse button, then right-click on it and select *Properties*. You can also view the HTML documentation, as well as open a window with the class declaration. To edit the documentation, go to the text layer window and edit the documentation annotation directly.

Each time you switch the focus from an class editing window (textual or graphical) to another class editing window, or you click on the check or simulate buttons, the content of the window is saved to a temporary file and passed to OMC. If there are syntax errors, then OMC cannot accept the content (because it cannot be parsed), and you get an error message. The content is then not saved, until you fix the syntax errors. If you are in a hurry, and need to save the content of the window, you can still go to the textual layer window, select Save As, and save the content as a text file somewhere. You can then exit SimForge without saving through OMC. When you later re-open the project, you can open the text view of your class, open the file you saved with another text editor, copy the entire content from the text editor and paste it into the SimForge text layer window. You can then resume debugging, until the syntax error are removed.

During editing sessions, you can permanently save your modifications by selecting the File|Save menu option on the text layer window, or by closing the window and confirming that you want to save. If you don't confirm, all the modifications since your last explicit load/save operation are lost. This is a very primitive Undo feature for your latest modifications - in the future, it is planned to add real Undo/Redo functionalities to SimForge.

If you want to work on a single class, then it is recommended that you open the text and graphical layers under the *Modelica classes* node of the project tree. You can then get the textual and graphical views, check the syntax, and simulate the model, all through OMC that runs in the background. However, in some cases you might want to directly edit a .mo file. In this case, you can open it under the *Modelica files* node - please note that this is just a text editor with basic syntax highlighting, which does not interact with OMC, and thus offers no syntax checking or graphical

views. Please also note that if you modify a .mo file using the .mo editor and the resulting file is not syntactically correct, you might later run into trouble, since OMC will not be able to parse it, and therefore its content will no longer be available under the *Modelica classes* node.

When you simulate a model, it is sufficient to provide the start and stop time for the simulation, as all other parameters have reasonable defaults. The interface for simulation settings will be improved in the near future, as OMC's interface is made more general. The results will appear as a new red node under the Simulation results node (and saved in the Temp sub-directory). If you want to keep the result, right-click on the node and select *Move to simulation results*. You can set a new name for the file, which will be stored in the Results sub-directory. The Temp sub-directory can then be safely cleaned every now and then, without losing important data.

Importing an existing Modelica package into a SimForge project for editing

If you have an existing Modelica package and you want to edit it using SimForge:

- create a new SimForge project, without adding anything. SimForge creates the directory structure and the Properties.xml file;
- close the project
- copy or move the Modelica package(s) you want to edit into the Modelica sub-directory of the project you've just created
- re-open the project
- go to Tools|Project properties and set all the required properties (version of OMC, version of Modelica Standard Library, location of other external libraries).

All the information that you have set with the Tools|Project properties are stored in the Properties.xml file. Note that this file does not contain any information on the content of the Modelica sub-directory, which is automatically scanned when the project is opened. This means that no explicit *import* operation must be performed. It is nevertheless necessary to set the paths of the Modelica Standard Library and of all the other required libraries, otherwise OMC doesn't know where to load them from.

Editing the Modelica Standard Library

When you load the Modelica Standard Library with the standard mechanism, the library is considered as read-only, so you can't change it. If you are a developer of the MSL, and you want to use SimForge to work on it, then you have to set up a project (see above), copy your local version of the MSL within it, and then you can work on it as if on any other user-developed package. It is also possible to include within your project both a custom version of MSL for development, and some other additional packages, e.g. to test the MSL.

Ladder to modelica converter

It is possible in SimForge to convert a ladder logic program into a modelica model.

This feature was implemented in order to let you create the models you want to simulate in modelica, then the control program in ladder (or sfc) and be able to simulate them all together.

Simforge provides two different controllers with different features and aims:

- Single ladder file converter
- Ladder project converter

The main difference between the two converters is that the project converter handles the shared variables between the various controllers providing a mechanism for variables buffering and refreshing.

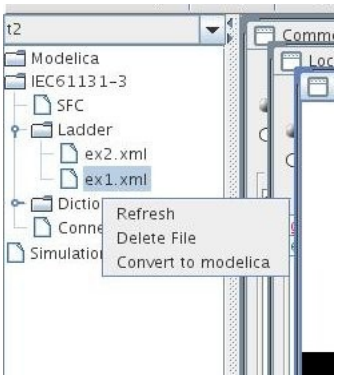
For more informations about the shared variables handling read the thesis: *An integrated opensource Modelica enviroment for PLC programming by Bruno Scaglioni (a.scaglioni_at_tin.it)*

Single file conversion

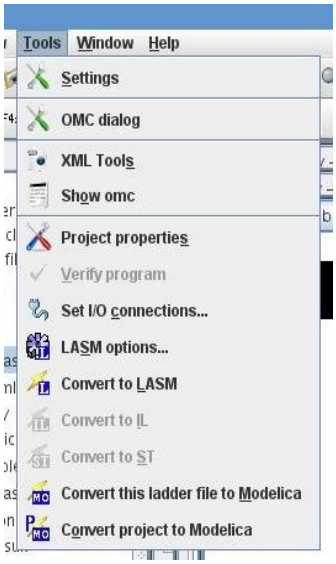
The single file converter can be reached in three ways, the first is the popup menu that appears when clicking with right mouse button on a xml ladder file in the tree on the left.

The second is by clicking on the first button shown below on the ladder toolbar.

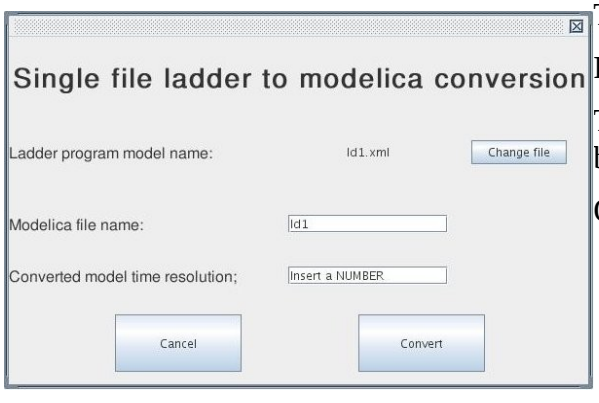
The third is by clicking on the menu entry in tools menu.



Popup menu in file tree



All these methods open the same dialog which is actually the one who controls the conversion.



This dialog presents some fields the user must fill.

First is shown the name of the ladder file to convert.

The user can change it by clicking on the **Change file** button. This action opens a File choosing dialog.

Once the user have choose the file to convert a

modelica name must be given to the controller, this will be the name of the modelica model.

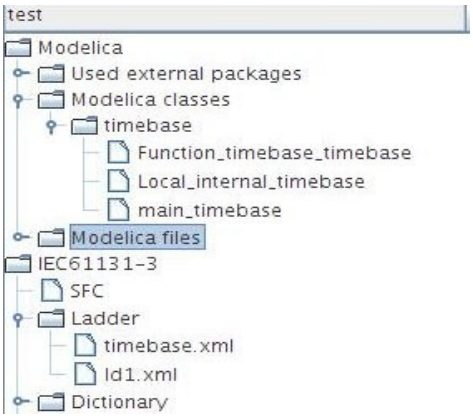
The field following the “Sampling time of the resulting Modelica algorithm” bust be filled with a decimal number.

This number is the sampling time of the modelica model representing the controller. This should follow the sample time of the real PLC machine to be simulated.

When pressing the **convert** button the conversion starts.

If everything works you should receive a message confirming the successful conversion.

After conversion You will find a new package containing three models.



The first model is the modelica function that actually does the calculations.

The second model is just a data structure that wraps the local variables.

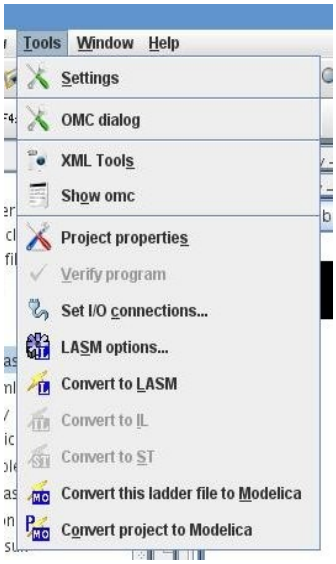
The third model is the container for input and output variables and for algoritmic code, this model is the only one that should interest the user. It is the one that the user should use in the simulation.

When the user wants to use the built modelica model the only thing to do is to use the drag and drop feature of Simforge modelica enviroment and to drop the third model discussed here into the simulation main model.

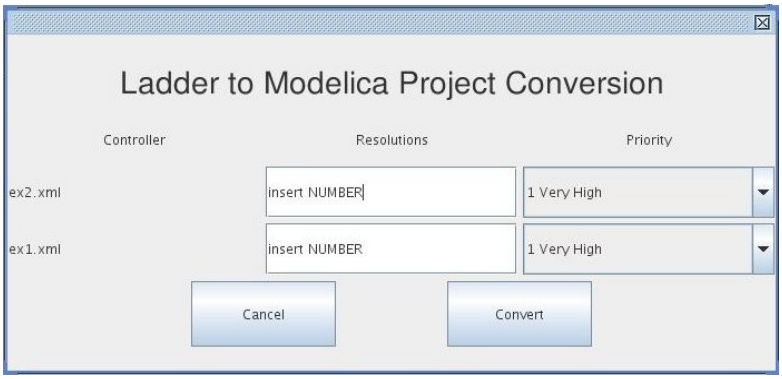
Project conversion

The project conversion dialog can be reached in two ways, the user can click the toolbar button on the simforge toolbar or it is possible to access the converter by the **tools** menu.

The button on the toolbar regarding the project conversion is the last one (the one with P letter on the top)



Once opened the project conversion dialog the screen is this:



This dialog is dynamically created calculating the number of controllers in the project at the time of the conversion.

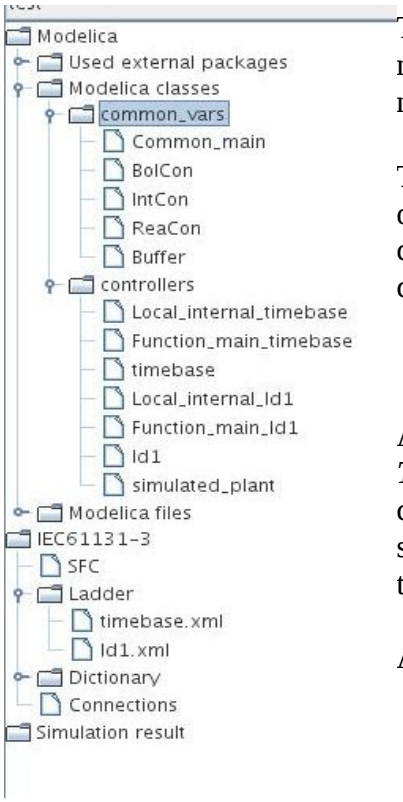
Basically for each controller the user must choose a name and a priority class.

The priority class is necessary for the correctness of shared variable mechanism, when converting a unique sequential number is given to each controller, the number is assigned with this mechanism:

Starting with the most high level of priority chosen by the user each priority class is analyzed, inside every priority class the number is given following a alphabetic order, when the number has been assigned to every controller in first class the converter assigns the following number to the first controller of the second priority class and so on....

This mechanism creates a priority list useful to determine what to do if two controllers write the same variable at the same time.

Once the conversion is complete the user will find two modelica packages in the modelica classes tree.



The first package called ***common_vars*** contains the shared variables models, the handler called ***buffer and*** some connector models necessary to handle the variables.

The second package contains the function models that calculate the output of every PLC clock cycle, the local variables modes and the controller models, which assume the same name of the ladder controllers.

A special model is the ***simulated_plant*** model which is the model ***TO SIMULATE!*** If the user wants the variables to be handled correctly the ***simulated_plant*** model is the one that must be simulated. In this model are contained all the connections that permit the buffer to share the variables.

All user-made models must be inserted in here.

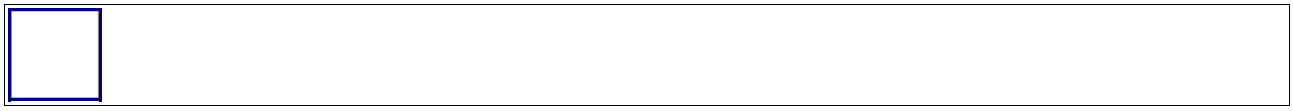
Regarding input / output variables they are allowed in project conversion, but they make sense only if the converted system will be inserted in a bigger system.

Simulated_plant model carries also graphical informations and connectors.

This part of the document is referring to AutoEdit that is the project SimForge has been derived to.

Some definitions and images need to be updated.

AutoEdit

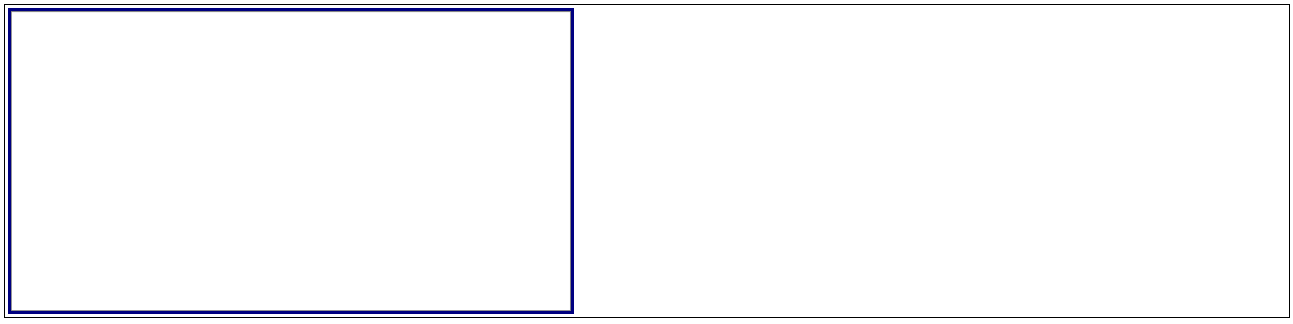


Efficient and flexible control software environment for IEC1131-3 standard. AutoEdit project started at Politecnico di Milano sede di Cremona on year 2004 and is actually under development.

Table of contents:

1. **Overview**
2. **User Manual**
3. **Authors**
4. **References**

Overview



Auto Edit is a PLC-oriented (Programmable Logic Controller) graphical software environment to design IEC1131-3 standard compliant logic control programs. The major advantage of GUI environments is that they make computer operation more intuitive, and thus easier to learn and use, because the actions are performed through direct manipulation of the graphical elements. With this application the user is allowed to directly write the control program as a simple circuit, translate it into LASM (Lego Assembly) language and upload the translated programs from the PC to the RCX Lego® Mindstorm.

User Manual

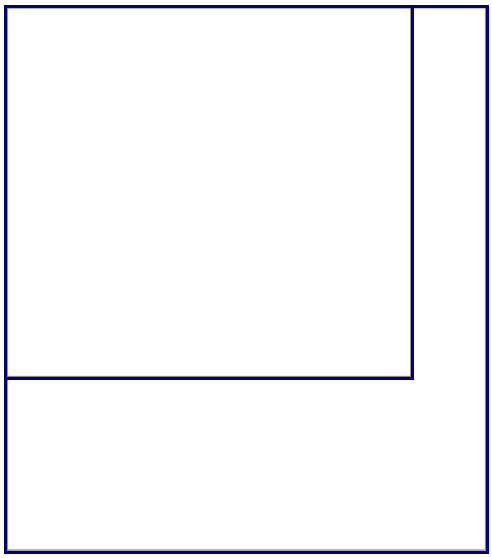
In the follow is reported the first version of the AutoEdit User Manual, please if you will find any error report it.

Application overview

AutoEdit is a graphical editor that supports both sfc and ld program (compliant to standard IEC1131-3) implementation, it has been designed to write logic control-oriented programs. The editor has many features, for example it's possible to check the presence of errors in the sfc diagram program, translate sfc program to obtain an equivalent ladder diagram program, compile it to obtain LASM language program, ...
AutoEdit uses the xml language to save the control program information; in this way the user can write directly the xml file and then he can view the content with AutoEdit xml viewer. Xml, stands for eXtensible Markup Language, XML provides a text-based means to describe and apply a tree-based structure to information.

Requirement, run the application and create a new program

In order to use AutoEdit it's necessary to have jdk1.5.0 or a most recent version of the jdk. You can download the jdk1.5.0 for free at http://java.sun.com/javase/downloads/index_jdk5.jsp .
On Windows OS start the command prompt, move to AutoEdit folder (where AutoEdit.jar is located) and then write the command line "java -jar AutoEdit.jar". The application will start automatically, otherwise, if environment variable are correctly setted, it's possible to directly double click on the icon.
With Linux OS the operations are the same: start the shell, cd to AutoEdit folder and then start the application with this command line: "java -jar AutoEdit.jar".



In particular, using Windows OS, you must open folder AutoEdit, select tools from menu, select folder Option, select the jar's extension in menù Files Types change the path "C:\Programmi\Java\ jre1.5.0_09\bin\javaw.exe" -jar "%1" %*" in ""C:\Programmi\Java\jre1.5.0_09\ bin\java.exe" -jar "%1" %*"; and close all.
Now you can run AutoEdit with double click on AutoEdit.jar icon. This manner is faster but not recommended, because you cannot read the output messages written in the standard output (in this case the shell). These messages are not fundamental to use the application but they can be useful to better understand the errors.

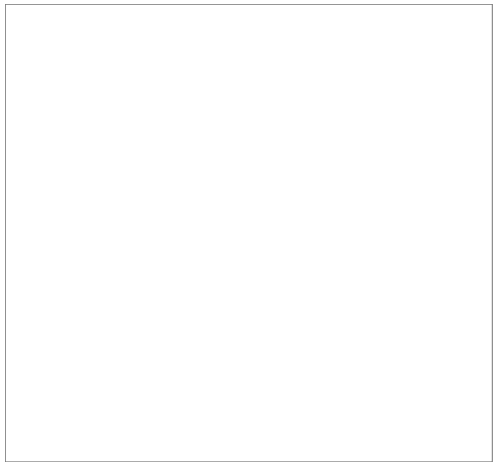


Fig. 1.1: the figure shows the modifies needed to directly open a file.jar on Windows OS

When launching AutoEdit you first view a window like that represented in follow figure. Now you can create a new file or open an old (previously created).

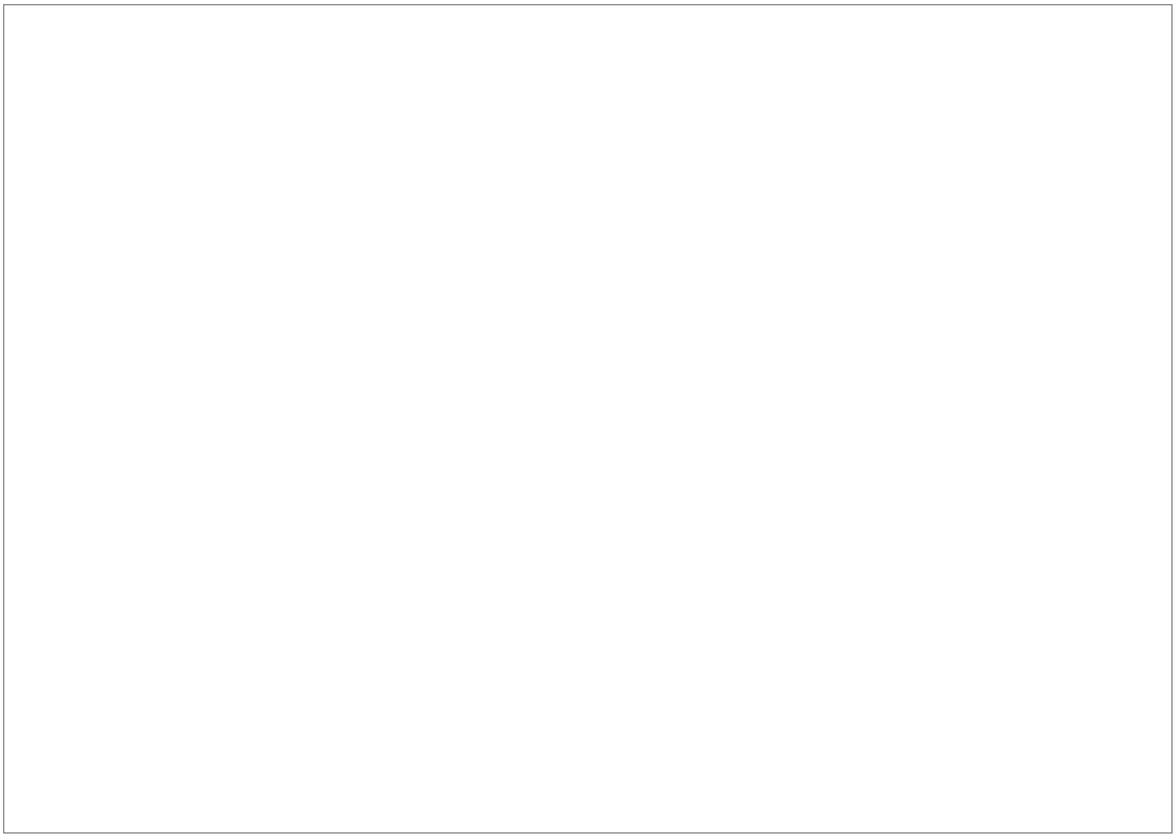


Fig. 1.2: the first window of the application

Create a new file

To create a new program select File -> New File. Alternatively it's possible to click on the icon : a "Create File" dialog box will appear. This dialog asks for the desired program's type you are going to create. By default ladder diagram is selected. The example here presented shows the sfc program's creation, so you have to choice Sequential Functional Chart radio button, as in picture below.

Fig. 1.3: the figure shows a screenshot of the dialog box to create a new file, the selection of the program type and the filename

Then you have to check that the selected folder is the folder where you want to work and specify the file name, in the example firstTest is the selected file name.

Remember that you must save the folder xsd in the working directory!!!
If you don't do it the application cannot run!!!

After this, you can start editing your first sfc program; the graphical environment is the follow picture:

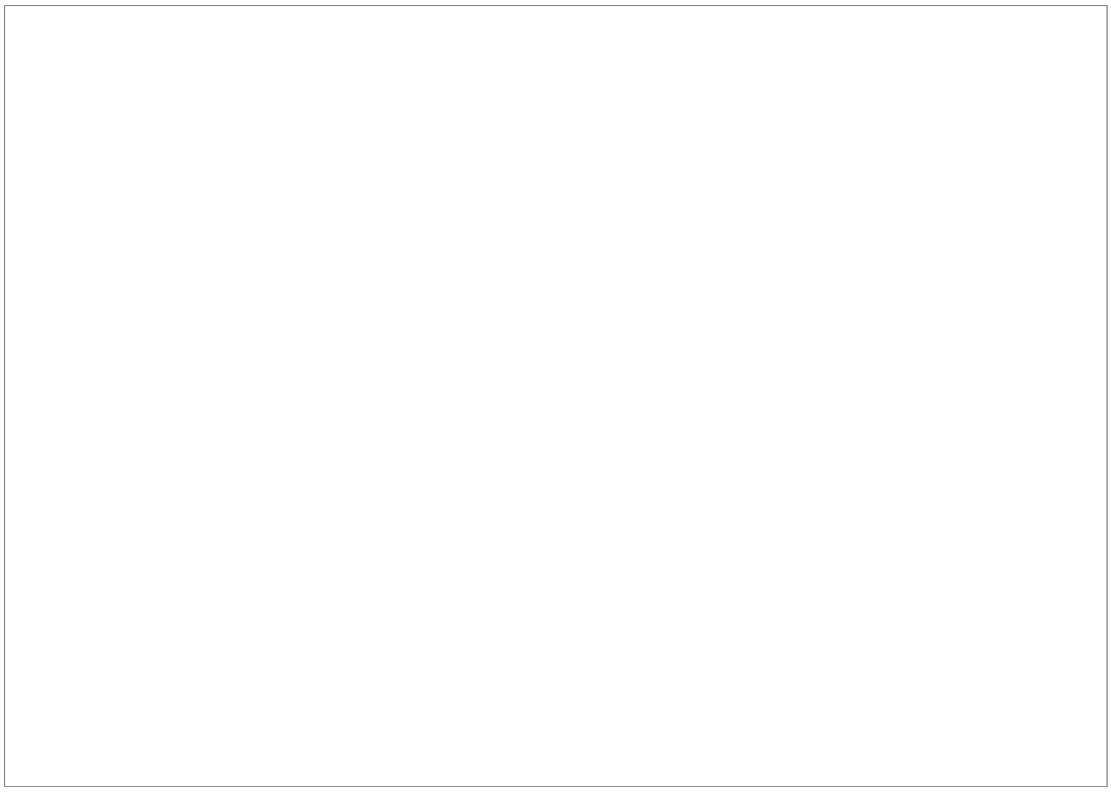


Fig. 1.4: New file creation dialog box

When you choice to create a file (or open a previously created file) the toolBar and componentBar will appear. Now it's illustrated the correct use of the toolbar and componentbar icons (for a more

satisfying explanation see the relative paragraph):

In toolBar there are the follow icons:

1. create a new file
2. open an existent file
3. save current diagram as xml file
4. activate/deactivate the dictionary visualization
5. show xml file of actual ladder diagram
6. choice the control checking settings for sfc diagram
7. convert sfc diagram in ladder

In toolBar the following icons are present:

1. insert a vertical line in current cursor position
2. insert a initial step in current cursor position
3. insert a step in current cursor position
4. insert a transition in current cursor position
5. insert a jump in current cursor position
6. insert a open convergence in current cursor position
7. insert a close convergence in current cursor position
8. insert a open parallel in current cursor position
9. insert a close parallel in current cursor position
10. insert a macro step in current cursor position
11. insert a vertical line in current cursor position
12. insert a final macro step in current cursor position
13. insert one empty row in current cursor position and move down to the follow line
14. insert one empty cell in current cursor position and move down to the follow line
15. delete the selected cell and move up
16. show xml file (the current file must be saved)
17. delete the selected cell

FirstTest it's a simple sfc program: it has an initialization phase, a transition, a macro-phase, another transition, a phase, a transition and a jump.

To create this diagram you have to move the cursor to the desired cell, use the mouse with "drag&drop" of the desired element or use the keyboard: on all the icons the correspondent key is reported (in this case the correct sequence is F2,F4,F10,F4,F3,F4,F5) and repeat the inserting procedure.

In firstTest there is a macroStep: you must create a simple expansion by creating an initial step macro, a transition and a final step macro.

Now on your screen you must see a diagram like this:

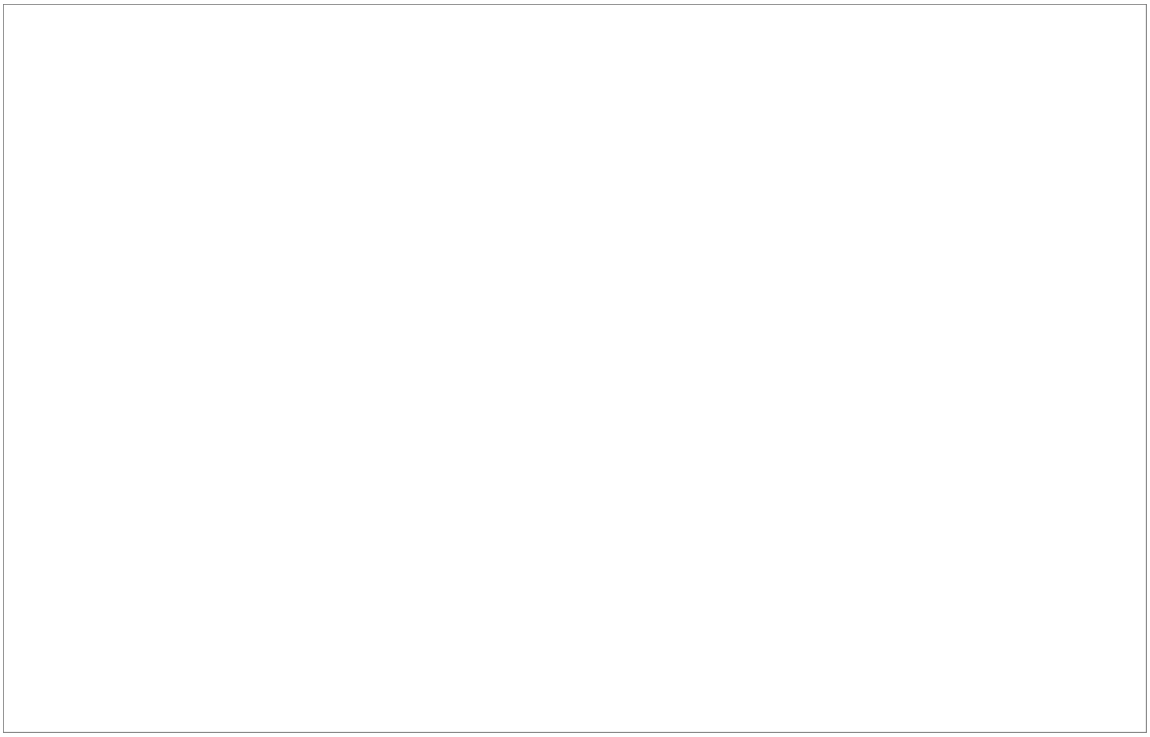


Fig. 1.5: The created diagram

You drawn a jump, a macroStep and its expansion. Now you must specify where the jump jumps, and which is the macroStep expansion. You can also associate a label to step and initial step. To do this you must place your cursor in the little square near step, as shown in the follow figure.

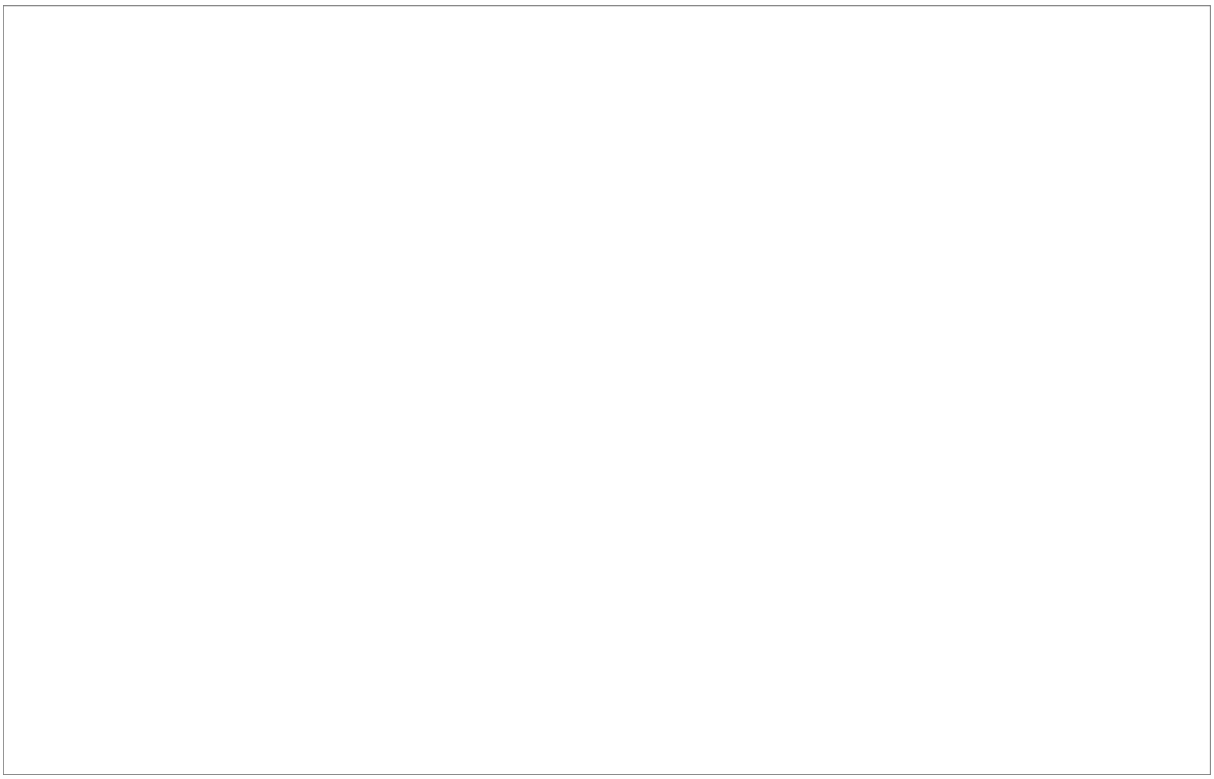


Fig. 1.6: Insert a label

When a jump is present you must indicate the correct jump label and assign the corresponding label to the step where you want to jump. For this reason the label must be unique in the diagram program.

Remark that the macro step, transition and jump can't have jump reference.

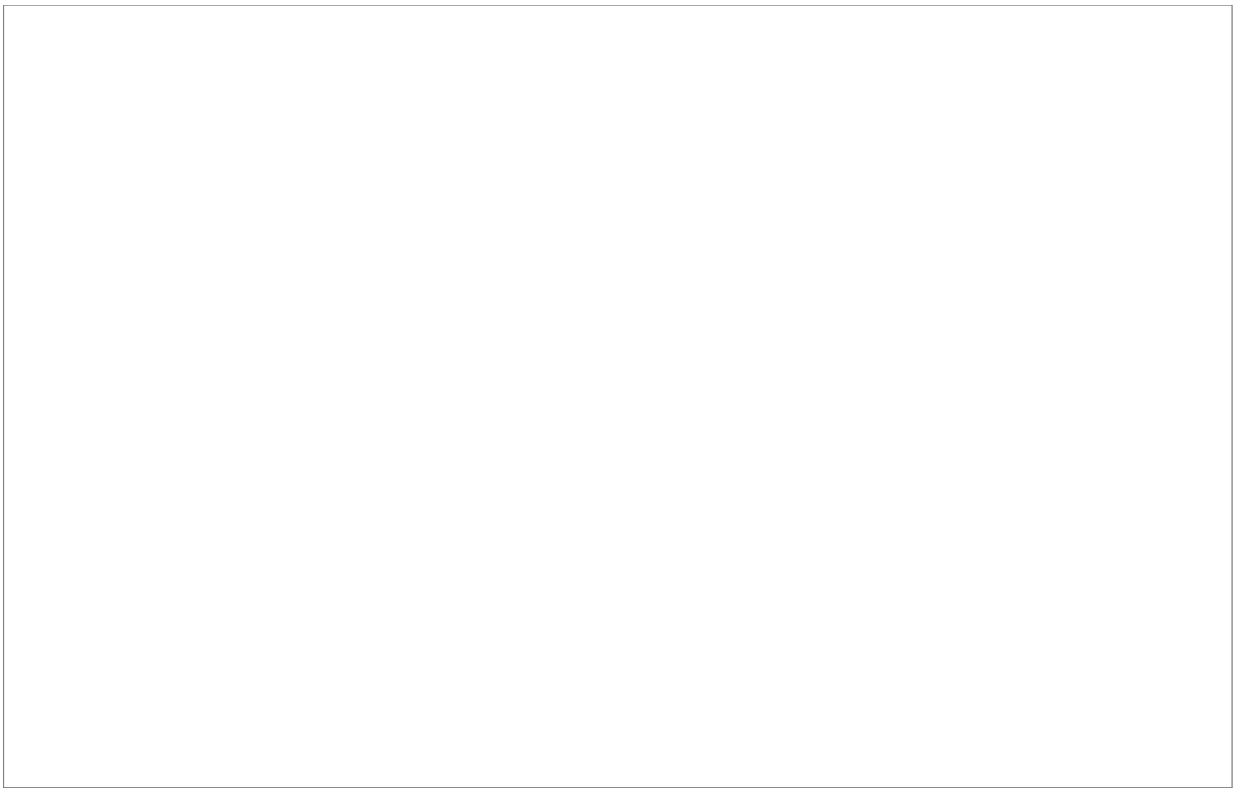


Fig. 1.7: jump use: "jump7" jumps to "init" labeled step

You can also insert the condition in the transition and the action in the step:

condition:

- you must put the cursor in the transition and write the desired condition. It's strongly recommended to save in the dictionary the used variable; it's not indispensable and unsaved variables is reported as an error only if you convert your sfc diagram to ladder diagram, but it's a good idea to do it.

action:

- the correct grammar rules to create new actions are reported in the follow lines:
  - S->A|B
  - A->nameAction (SET) if it's a set action
    - nameAction (RST) if it's a reset action
    - nameAction (NET) if it's a net action
  - B->A;B

As you can read, if you want write several actions you must use a semi-colon separator and write them on the same row. To write the action put your mouse in the text cell in front of the step. Remember that you cannot associate action to macro step: this is a non sense. If you want to do an action you must associate it to a step in the macroStep's expansion.
In the text area in front of the macrostep you must specify the initial step and final step of correspondent expansion separated by "-". Two macroSteps can't refer to the same expansion. In the follow figure you can view that "step5" has an action like the "Macro 13" initial step, that the "transition4" has a condition and that in the macroStep is specified the expansion.

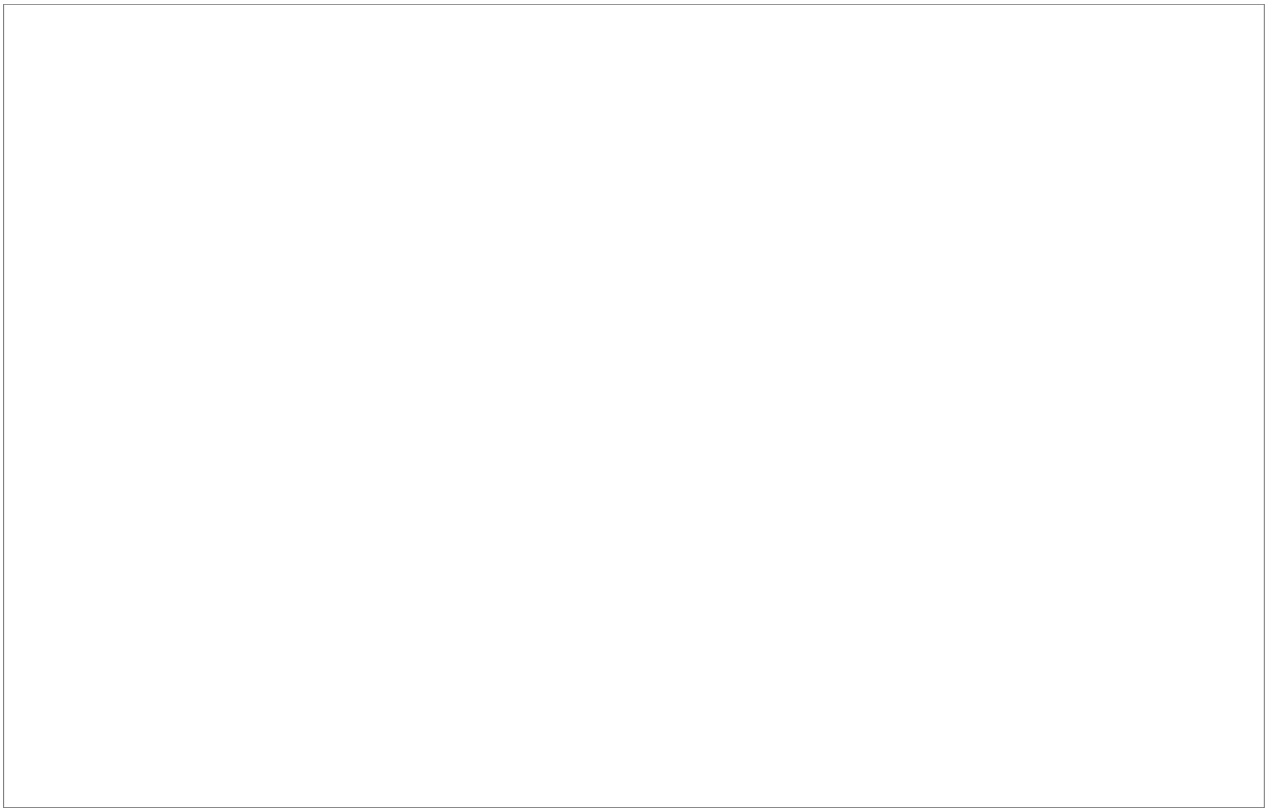


Fig. 1.8: insert action, condition, macroExpansion

REMARK:

- *In order to use AutoEdit you must save the project files in the xsd folder.*
- *It's not possible to use special characters (non UTF-8) for filename.*
- *The variable name must contain at least one alphabetic character*
- *jump label must refer to phase label and not to label ID.*

Now firstTest diagram it's complete. It's possible to save it, translate it into ladder or close the application without saving nothing.

Save the diagram and read the xml representation

As implemented in the current version of the applications it's possible to save the file in different way: by simply clicking on the icon , and by choosing File->Save from menu or Save As.
If you are choosing Save from the Menu a dialog box asking filename and location will appear.

When the ion is clicked the file is automatically saved .

It's possible to read xml file of the current diagram by clicking on the icon

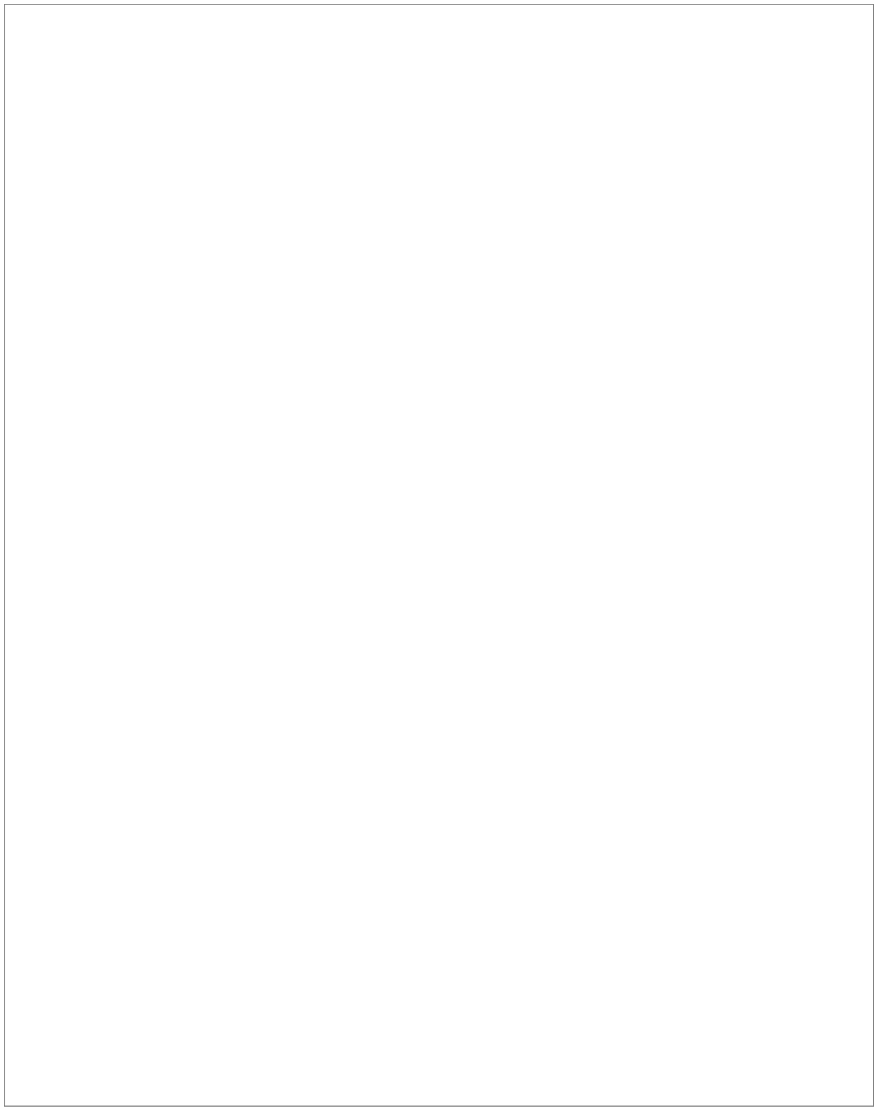


Fig. 1.9: xml representation of the current diagram

Remark: The xml file exists only after the save; therefore when you want to read xml file you first have to save diagram.
Remark that the xml description is not only a topological description: it's a semantic description. The xml contains position information about the element but also network properties information, for example if and where a parallel is present.

Conversion from sfc diagram to ladder diagram and xml ladder viewer

It's possible to convert the sfc diagram into an equivalent ladder schema.
First you must verify the program correctness. The user can select the desired controls: by clicking on the icon it's possible to choose base or customized controls. The base control selection grants the IEC1131-3 program compliance.

Also by choosing customized controls the IEC1131-3 program compliance is assured.

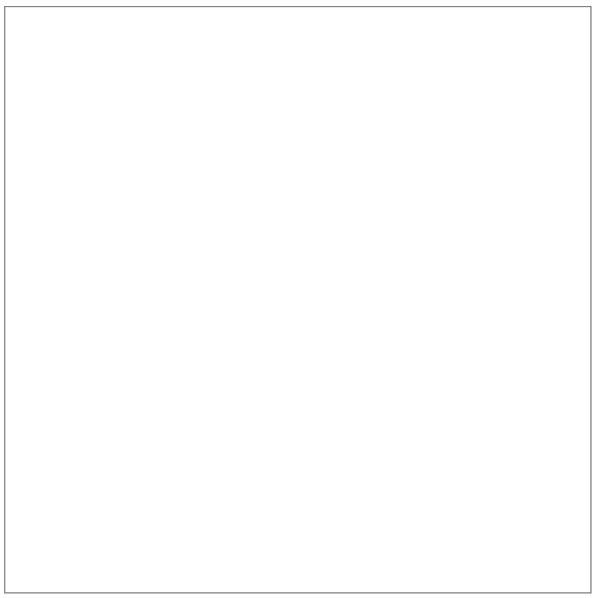


Fig. 1.10: control selection

After the selection of the desired control it's possible to translate the program into ladder by clicking on the icon and a dialog box will report the translation output.

It's possible to read the xml description of the ladder diagram you have just created in the ToolBar.

To view ladder schema of the translated file you must open this file from the MenuBar. File-> Open shows a dialog box where you choice the file that you want to open, in this case firstTest_LD. The postfix _LD is add to the fileName.

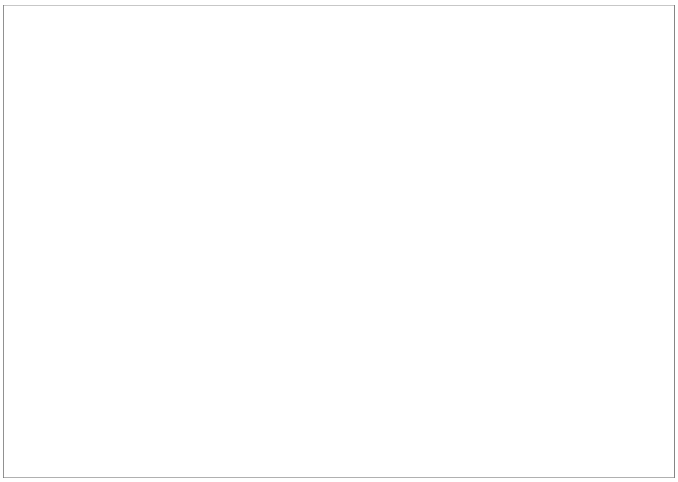


Fig. 1.11: Open file

It's possible to view ladder schema.

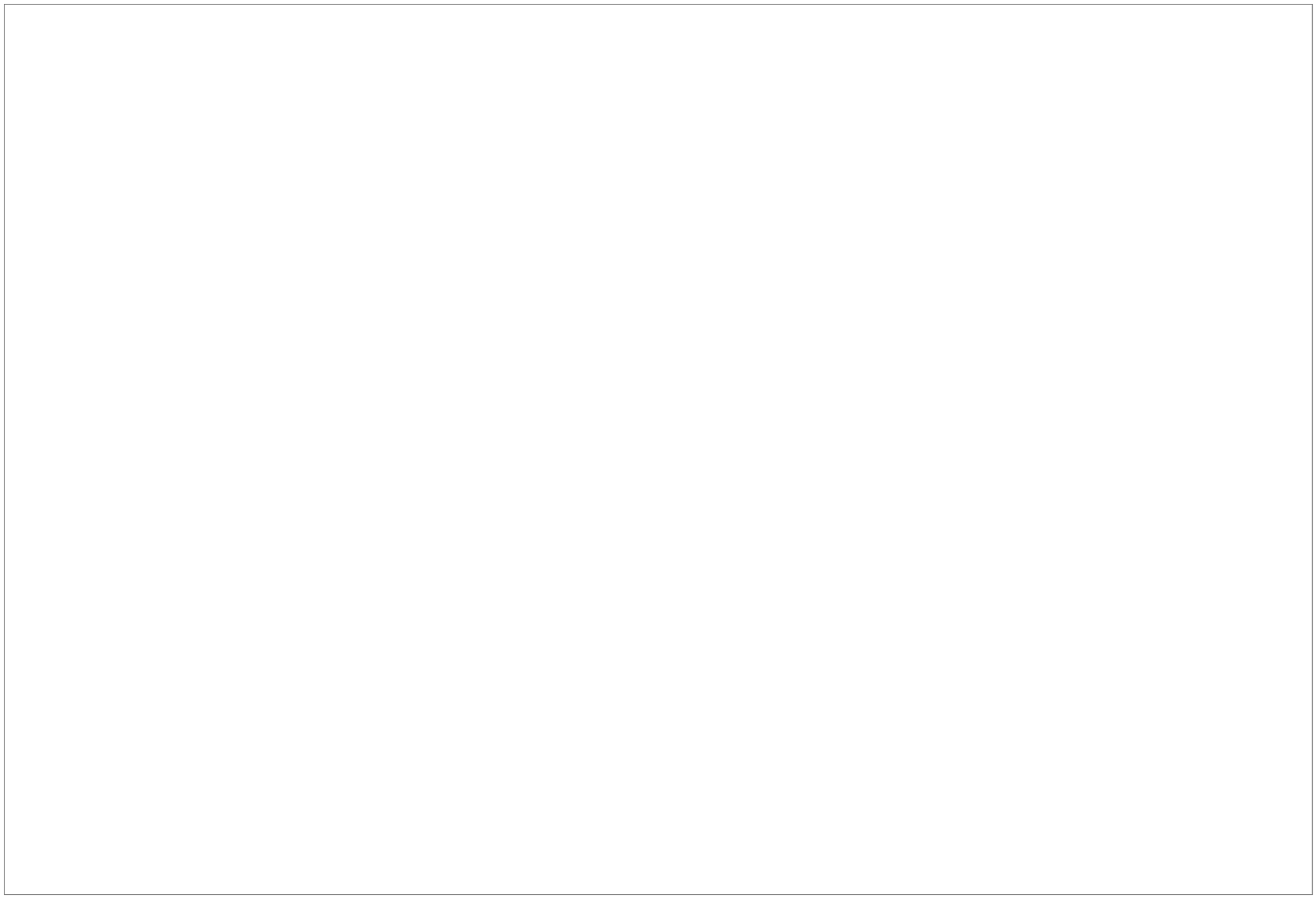


Fig. 1.12: "firstTest_LD" ladder diagram obtained from "firstTest" sfc translation

Remark that to translate the sfc diagram into a ladder program is necessary to create system variables (declared as local variable). This happen for the sfc step, transition, arithmetic and logic variables.

References

- Lego
- PLCopen for efficiency in automation